NPSCS-91-014

NAVAL POSTGRADUATE SCHOOL
Monterey, California

A STOCHASTIC APPROACH TO THE WEIGHTED-REGION PROBLEM :
I. THE DESIGN OF THE PATH ANNEALING ALGORITHM

Mark R. Kindl
Man-Tak Shing
Neil C. Rowe

June 1991

Prepared for:

Navy Center for Applied Research in Artificial Intelligence
Washington, D.C. 20375-5000

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared for the Navy Center for Applied Research in Artificial Intelligence and funded by the Naval Postgraduate School.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): NPSCS-91-014	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Navy Center for Applied Research in Artificial Intelligence
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5100	7b. ADDRESS (City, State, and ZIP Code): Washington, D.C. 20375-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: O & MN, Direct Funding
8c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A Stochastic Approach to the Weighted-Region Problem: I. The Design of the Path Annealing Algorithm (U)

12. PERSONAL AUTHOR(S)
M.R. Kindl, M.T. Shing, N.C. Rowe

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Techical	FROM 10/90 TO 9/91	1991 June 26	44

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	path planning, shortest path, simulated annealing, weighted regions

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
This paper presents an efficient heuristic algorithm for planning near-optimal high-level paths for a point agent through complex terrain modeled by the Weighted-Region Problem. The input to the Weighted-Region Problem is a set of non-overlapping convex homogeneous-cost regions on a two-dimensional plane. Each region is associated with a cost coefficient (or weight), which indicates the relative cost per unit distance of movement in that region by the point agent. The weighted distance between two points in a convex region is the product of the corresponding cost coefficient and the Euclidean distance between them. Given a start and a goal point on the plane, the objective of the Weighted-Region Problem is to find a minimum cost path from start to goal through the weighted regions. We have designed and developed a very efficient algorithm for finding near-optimal solutions for the Weighted-Region Problem using a combination of the classical AI heuristic search techniques and the probabilistic combinatorial optimization technique called simulated annealing. Extensive test results (to be presented in Part II of the paper) indicate that the new algorithm runs much faster than previous known techniques with a very minimal sacrifice in optimality.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Man-Tak Shing	22b. TELEPHONE (Include Area Code): (408) 646-2634	22c. OFFICE SYMBOL: CS/SH

A Stochastic Approach to the Weighted-Region Problem : I. The Design of the Path Annealing Algorithm

Mark R. Kindl[1] **Man-Tak Shing**[2] **Neil C. Rowe**[3]

ABSTRACT

This paper presents an efficient heuristic algorithm for planning near-optimal high-level paths for a point agent through complex terrain modeled by the Weighted-Region Problem. The input to the Weighted-Region Problem is a set of non-overlapping convex homogeneous-cost regions on a two-dimensional plane. Each region is associated with a cost coefficient (or weight), which indicates the relative cost per unit distance of movement in that region by the point agent. The weighted distance between two points in a convex region is the product of the corresponding cost coefficient and the Euclidean distance between them. Given a start and a goal point on the plane, the objective of the Weighted-Region Problem is to find a minimum cost path from start to goal through the weighted regions. We have designed and developed a very efficient algorithm for finding near-optimal solutions for the Weighted-Region Problem using a combination of the classical AI heuristic search techniques and the probabilistic combinatorial optimization technique called simulated annealing. Extensive test results (to be presented in Part II of the paper) indicate that the new algorithm runs much faster than previous known techniques with a very minimal sacrifice in optimality.

KEYWORDS:

path planning, shortest path, simulated annealing, weighted regions

---

(1) MAJ(P) Mark R. Kindl, USA is with the Army Institute for Research in Management Information, Communications, and Computer Science (AIRMICS), 115 O'Keefe Building, Georgia Institute of Technology, Atlanta, GA 30332. Research reported here was done while MAJ Kindl was with the Computer Science Department, Naval Postgraduate School.

(2) Man-Tak Shing is with the Computer Science Department, Naval Postgraduate School, Monterey, CA 93943. This paper was prepared for the Navy Center for Applied Research in Artificial Intelligence. Funding was provided by the Naval Postgraduate School.

(3) Neil C. Rowe is with the Computer Science Department, Naval Postgraduate School, Monterey, CA 93943.

1. INTRODUCTION

This paper presents an efficient heuristic algorithm for planning near-optimal high-level paths for a point agent through complex terrain modeled by the *Weighted-Region Problem* (WRP) [18]. The input to the Weighted-Region Problem is a set of non-overlapping convex homogeneous-cost regions on a two-dimensional plane. Each region is associated with a cost coefficient (or weight), which indicates the relative cost per unit distance of movement in that region by the point agent. These cost regions are assumed to be isotopic, so that their respective weights are invariant with respect to the direction of movement. Examples of cost measure include ease of movement through terrain, cover and concealment, exposure to enemy positions, fuel or energy consumption, lethality of enemy weapons, or areas of contamination [31,36]. The weighted length of a path segment joining two points in a convex region is the product of the corresponding weight and the Euclidean distance between them. A path segment which coincides with a *boundary edge* (i.e. an edge separating two weighted regions) incurs the lesser weight of the two regions separated by the edge. Given a start and a goal point on the plane, the cost of any path joining the start and goal is the sum of the weighted lengths of all segments forming the path. The objective of the Weighted-Region Problem is to find a minimum cost path from start to goal through the weighted regions.

Due to the complex solution space, WRP cannot be solved with standard continuous-optimization techniques, e.g. draw a line from start to goal and then perturb the line until local optimum is reached. Most of the existing algorithms solve the WRP by means of combinatorial searching techniques. One way to tackle the WRP is by means of the Wavefront Propagation technique. The Wavefront Propagation technique is an approximating method which overlays the input map with a grid. It searches through the grid from the start in a breadth-first manner until the goal is reached. The first path that reaches the goal is retrieved as the solution. The algorithm has two drawbacks. First, even a simple map with a few regions may require a high resolution grid to accurately represent the detail. As a result, execution times suffer. Second, the requirement that all paths be fixed to the grid introduces digital bias. This stair-stepping effect results in solutions that are as much as 8% longer than the optimal solutions [25]. Attempts to smooth paths may result in paths worse than 8%, or make feasible

paths infeasible.

In [16, 17], Mitchell and Papadimitriou presented a continuous Dijkstra's algorithm that finds an optimal solution for the WRP. The algorithm runs in $O(n^7 L)$ time using $O(n^3)$ space, where n the is total number of vertices of the regions and L is the precision of the weights in the problem instance. Rowe and Richbourg [30] also developed a Snell's-Law-based A^* Search method to find an exact solution. Although their algorithm has an exponential worst-case time complexity, they showed empirically that his Snell's-Law-based approach performs better than the continuous Dijkstra's algorithm on the average.

When applying the WRP to real-life problems, it is often necessary to combine weighted-region overlays into more complex maps in order to plan paths that simultaneously optimize over several mission-critical parameters. For example, by weighting the relative importance of each cost measure, an optimal solution on a composite of three maps could represent the path of combined minimum travel time, at least exposure to the enemy and with maximum survivability [7]. Difficulty of optimization increases with the number of overlays. As problem instances approach the reality and complexity of battlefield mission planning, faster solutions become more desirable than absolute optimality. Existing algorithms are too slow for planning routes in a complex battlefield. In order to speed up the search process, we propose to tackle the WRP by means of stochastic search. The proposed algorithm, called *path annealing*, is based on our customization of a probabilistic combinatorial optimization technique known as *simulated annealing* [13]. Simulated Annealing has been used successfully in finding near-optimal solutions to problems having unusually large and complex search spaces. Clearly, an algorithm which uses simulated annealing cannot guarantee a globally optimal solution. However, experiments conducted by other researchers [4, 13, 26, 27] do show that simulated annealing based algorithms can generate near-optimal solutions in reasonable execution times for very complex search spaces.

There are several good reasons for tackling the WRP using simulated annealing. From a computer science viewpoint, the search space of the WRP appears to be well-suited for such an approach. Depending on the resolution required by a particular application, a problem instance could be

arbitrarily complex in terms of the number of regions, the number of vertices and edges composing them, or their shapes and orientations. The approximating nature of the polygonal regions, coupled with the sensitivity of the true optimal paths to very small changes in geometry and coefficient magnitude, suggests that the effort associated with finding a true optimal path is rarely worth the result. Like all hard optimization problems with complex solution spaces, WRP tends to have a relatively large number of local minima with costs that are very close to the globally optimal solution(s). The simulated annealing technique takes this as its underlying assumption, and settles on one such solution within a reasonable amount of time.

Solutions generated by the simulated annealing approach are also appealing from the military commander's perspective. A globally optimal path may be undesirable since it is usually a very exclusive solution. Surely, if we can find the set of globally optimal paths for a given problem instance, then the enemy can do likewise! Although the possibility exists that our algorithm will discover an optimal solution, it generally finds one of many near-optimal paths. Such paths do not necessarily lie spatially close to the optimal (an obvious tactical advantage). Furthermore, the quality of the solution generated by the annealing process improves with time. For applications with uncertain time constraints, the annealing process can be halted prematurely to return the "best" solution to date.

Moreover, the annealing approach to solving the WRP is interesting as well as unusual for two reasons. First, unlike most problems to which simulated annealing has been applied in the past, the WRP is solvable in $O(n^7L)$, where n represents the number of map vertices and L the number of bits precision [18]. Most, if not all, annealing research has been applied to problems which are NP-Hard. Hence, it is a real challenge to design a good annealing-based solution to the WRP. Second, although simulated annealing are often criticized for its large running times and its inability to compete with problem-specific heuristics [19], our empirical studies indicate otherwise. Our approach to the WRP attempts to merge simple intelligence with practical engineering by coupling simple heuristics to non-exhaustive, stochastically controlled sampling. Path annealing, though primarily a stochastic algorithm, makes important use of A* search. In fact, it is fair to say that path annealing draws a major part of its performance and power from its A* search component. Rather than just another application of

simulated annealing, path annealing is an intelligent marriage of global and local search (both continuous and discrete) under probabilistic control.

We employ simulated annealing as a simple tool for the control of local search. In doing so, many of the specific design concepts (e.g. the annealing schedule) are adapted from previous research. Therefore, we emphasize that the significant contribution of our work is not in annealing, but rather in the novelty and design of our annealing-based approach to solving the WRP, as well as the set of heuristics and bounding techniques to increase the performance and efficiency of the proposed algorithm.

We have implemented path annealing in Quintus Prolog Version 2.5 (w/ ProWindows Version 1.1) and C on a Sun-4 (SPARC) Workstation running SunOS 4.0. This paper presents the design of the algorithm. Summary of the performance enhancement techniques and the results of the extensive empirical study will be presented in Part II of the paper.

2. The Basic Simulated Annealing Algorithm

Simulated annealing is an adaptive search technique based upon the Metropolis Algorithm [15], which simulates a complex system of particles (molecules) in a heat bath. Recognizing concepts similar to optimization, Kirkpatrick *et al* [13] and Cerny [5] independently developed simulated annealing. Since then, many researchers have used it to obtain solutions to a variety of combinatorial optimization problems. In its simplest form, simulated annealing works as follows:

Input:

(1) The solution space of the optimization problem.

(2) The control parameters for the annealing process, which include

(a) T_0 - the initial value of the control temperature T,

(b) T_f - the final value of T,

(c) R - the reduction factor for T (typically $0.70 \leq R \leq 0.99$),

(d) L - the maximum number of attempted moves at any each value of T,

(e) L_s - the maximum number of accepted moves at any each value of T.

Output:

An optimal or near-optimal solution.

Algorithm

Begin

Current_Solution := some solution from the given solution space;

$T := T_0$;

Best_Solution := *Current_Solution*;

while ($T > T_f$) do

begin

$N := 0$; /* tracks the number of moves attempted at the current value of T */

$N_s := 0$; /* tracks the number of moves accepted at the current value of T */

while (($N < L$) and ($N_s < L_s$)) do

begin

/* perturb current solution randomly to obtain a new legal solution */

New_Solution := **move** (*Current_Solution*);

$N := N + 1$;

ΔC := **cost** (*New_Solution*) − **cost** (*Current_Solution*);

if (($\Delta C \leq 0$) or (**random** ( ) $\leq e^{-\Delta C / T}$))

then begin

Current_Solution := *New_Solution*;

$N_s := N_s + 1$;

if (**cost** (*Current_Solution*) < **cost** (*Best_Solution*))

then *Best_Solution* := *Current_Solution*;

end;

end;

$T := T \times R$;

end;

Output (*Best_Solution*);

End.

Starting from the initial solution and its evaluated cost, the move generator randomly perturbs this solution to obtain a new one. The cost of the new solution is computed and compared to the current solution. As in the standard local iterative improvement approach, the new solution always replaces the current one if the change in cost, ΔC, is non-positive. However, unlike the local iterative improvement approach, the new solution is accepted with probability $P = e^{-\Delta C / T}$ if ΔC is positive.

The control temperature, T, is a value in the same units as the cost function. It regulates the probability distribution that defines the acceptance criteria of solutions with increasing costs. At each temperature, T, the procedure attempts up to L moves of which up to L_s acceptances are permitted. The temperature is reduced by a factor R, and both N and N_s are reset to zero. The resulting behavior is a downward-biased random walk through the solution space, with the ability to escape local minima. Gradually decreasing control temperature changes the exponential probability distribution. This tightens the acceptance criteria against larger cost increases. The value of T for which no cost increases are reasonably expected and no more improvements can be found is T_f, the freezing temperature. At this stage the algorithm returns the current solution, and halts.

There are a variety of heuristics, enhancements and improvements for the basic annealing algorithm. Refer to [6] for an extensive bibliography of simulated annealing theory and applications.

3. Path Annealing - Customization of Simulated Annealing to WRP

Without loss of generality, we make several assumptions regarding the input map. As in [18], cost coefficients are integers taken from the set $\{w, w+1, w+2, \cdots, W, +\infty\}$. If necessary, rational number coefficients can always be handled by rescaling. Furthermore, we assume that all weighted regions are convex. Even if this is not the case, it is always possible to insert additional boundary edges between existing vertices to make all regions convex. Such edges will become boundaries between regions of equal cost. We will refer to these as *phantom edges* because their sole purpose is to ensure convexity. *Border edges* and *border vertices* define the outer limits of the map, and bound the

only non-convex region. The current implementation of path annealing requires that linear regions (such as roads in [18] and [29]) must be modeled by regions of positive (non-zero) area. However, this restriction is not essential to the algorithm itself, and can be removed by a few data structure modifications. Five basic components are needed to adapt the simulated annealing technique to solve the Weighted-Region Problem:

1. Solution space definition,
2. S_0 - Initial solution taken from the space,
3. cost(S_i) - Cost function to be evaluated for each solution,
4. Move generator to randomly generate small changes in solutions,
5. An annealing schedule (i.e. the parameters T_0, T_f, R, L and L_s) to provide algorithm control.

4. Efficient Representation of the Solution Space

For annealing to perform efficiently, the search space must be discrete and finite. Furthermore, we would like to reduce the size of the search space as much as possible. Unlike other optimization problems to which simulated annealing has been applied, the WRP solution space is continuous, and hence, uncountably infinite. We impose discreteness by defining the concept of *window sequence* (WS).

We assign a unique integer identifier to each region boundary edge. Furthermore, we assign identifiers S and G to designated start and goal points respectively. We now define the concept of window sequence. This concept was originally adopted by [28] to describe a family of paths which cross the same sequence of boundary edges and vertices, and which must obey the same set of movement constraints. Our definition of window sequence is similar, but not identical. Consider the continuous, infinite set of all paths between the start and goal points which cross the same ordered sequence of edges. We can uniquely label this family of paths using an ordered list of edge identifiers. The first and last elements of this list are always S and G corresponding to start and goal respectively. The remainder of the list includes the identifier of every edge the associated paths cross. Such an ordered

list is called a *window sequence*. Figure 1 is an example of a window sequence. Duplicate edge identifiers are permitted within a legitimate WS. This simply means that a family of paths crosses the same edge multiple times. Partitioning the original continuous solution space of the WRP into WS's has several advantages:

1. Each WS has a discrete representation in terms of regions and edges.
2. The total number of WS's is finite.
3. Every possible path in the continuous space between start and goal is contained in some WS.
4. Some WS contains the globally optimal path.
5. The convexity theorem of [18] and [30] still applies to WS because the theorem places no restriction on edge length and the path-cost function remains the same. Therefore, a unique locally optimal path must exist within each WS, and a standard continuous function optimization method can be used to locate it.

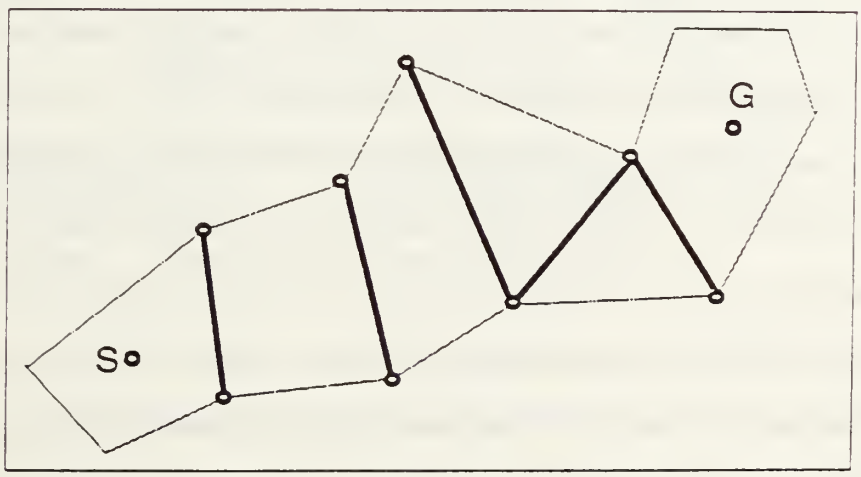

Figure 1 Window Sequence

By defining the cost of a WS as the cost of the locally optimal path within the WS, we have a means of comparing WS's. Hence, we can consider WS's as states in the annealing process and solve the WRP by searching for the optimal WS. In order to have an efficient high-level representation of the solution space, we preprocess a WRP map into several primitive data structures designed to facilitate rapid navigation through the solution space. Our input maps are just a list of boundary edges with defining vertices and associated cost coefficients to either side. It is assumed that this list represents a

complete set of straight-line segments in a planar graph. Henceforth, we refer to this graph as a *weighted-region map*. The input description of each edge in the map conforms to a standard convention. A boundary edge, E, is defined by the tuple (V_1, V_2, μ_1, μ_2), where V_1 and V_2 are its terminating vertices, and μ_1 and μ_2 are the cost coefficients of the regions it bounds. An agent traveling along this edge from V_1 to V_2 will always see μ_1 on his left and μ_2 on his right if $\mu_1 < \mu_2$. However, if $\mu_1 = \mu_2$, then the x coordinates of the vertices are related as $x_{V_1} < x_{V_2}$. If $\mu_1 = \mu_2$ and $x_{V_1} = x_{V_2}$, then $y_{V_1} < y_{V_2}$.

Map preprocessing creates and assigns a unique integer identifier to each edge, and stores it together with the defining vertices, associated weights, linear equation form, midpoint, length, bounded regions, and directional data. This provides extremely efficient access since Quintus Prolog [24] hashes predicates on the first atomic argument, in this case the edge identifier. Thus, given a WS of edge identifiers, its vertices, regions, and related data are all immediately accessible.

Since we desire quick access among edges, vertices, and regions, preprocessing also creates several data structures which are essentially inverted files indexed on these entities. For each vertex in the map, a *vertex edge-list* is maintained. This data structure stores a clockwise ordered list of edges incident around each vertex. Two similar data structures are constructed for regions. A *region edge-list* holds the associated cost coefficient and the ordered list of boundary edges for each convex region. A *region vertex-list* stores the ordered list of vertices around each region. As will be explained later, obstacles are treated as though they were special oversized vertices. Similar to a vertex edge-list, an *obstacle edge-list* stores an ordered list of the incident edges around each obstacle.

To facilitate the computation of the initial WS and the generation of new WS's during annealing, we also create a dual representation of a WRP map. A connected graph embeddable in the Cartesian plane without intersecting arcs and whose vertices are of minimum degree two is known as a planar graph. Restricting the arcs of a planar graph to straight lines results in a planar straight-line graph (PSLG) [22]. Our WRP input maps are PSLG's. We further restrict these maps to consist of convex regions only. For all such structures, graph theory defines the notion of a dual graph. Regions in the planar graph become vertices in its dual graph; whereas vertices in the planar graph become regions in its dual graph. In the dual graph, arcs connect vertices corresponding to adjacent regions in the original

graph. Thus, a one-to-one correspondence exists between arcs in the planar graph and arcs in its dual. While this kind of dual graph is often a valuable problem transformation (e.g. Min/Max Cut Problem), it is not that useful for our purposes. Instead, we construct a different kind of dual representation for a WRP map, called the *edge dual-graph*, as follows.

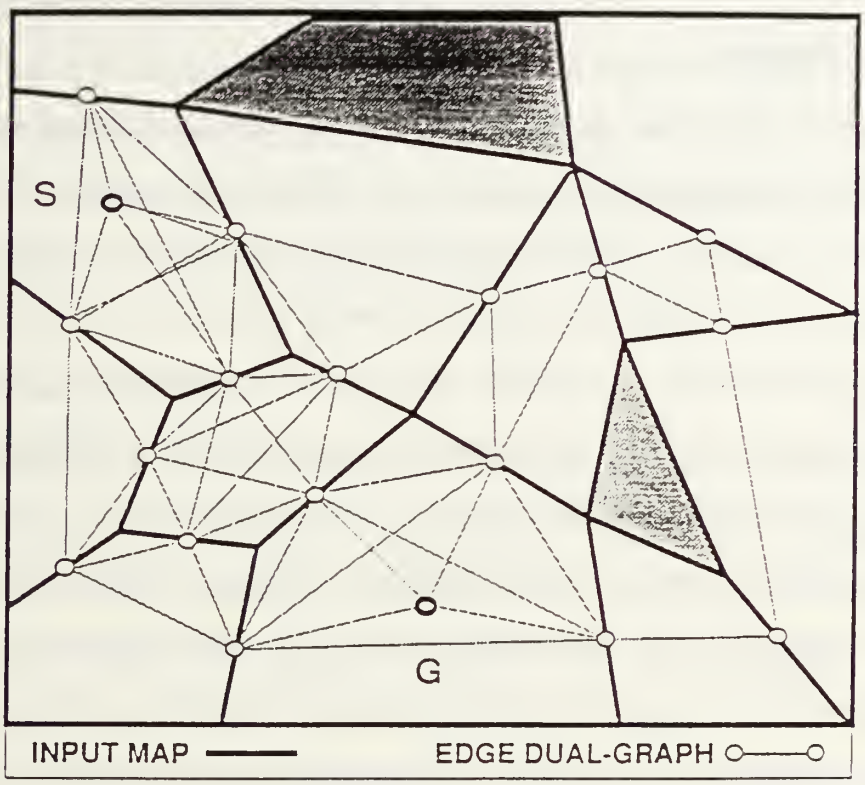

Figure 2 Weighted-Region Map and Edge Dual-Graph

The discussion which follows refers to two distinct representations of a problem instance - the weighted-region input map and its corresponding edge dual-graph. In order to avoid confusion, we will use the terms *vertices*, *edges*, *regions*, and *map* to describe a weighted-region map. We will use the terms *nodes*, *arcs*, and *graph* to describe the edge dual-graph. The edge dual-graph is essentially an adjacency list representing the spatial structure of the map at a relatively high level of abstraction. To create this graph, we assign a node to the midpoint of each map edge which does not bound an obstacle (or the border). Special nodes are assigned to the start and goal points. In each non-obstacle region, we add arcs to connect all nodes at the midpoints of the edges which bound the same region. The fact that all regions are convex guarantees that all such arcs cannot intersect obstacles or other regions. Thus, each region contains a clique consisting of all nodes representing its non-obstacle

boundary edges. The regions containing the start and goal points will have additional arcs connecting the start or goal node to the clique. Figure 2 illustrates a simple WRP map with its associated edge dual-graph.

5. The Initial Solution

Annealing can begin from any solution in the space. However, empirical evidence suggests that reasonably good initial solutions can often provide better final results [10, 20]. The obvious straight-line path from start to goal is not always feasible when obstacle regions are present. While there are good algorithms for constructing and searching a visibility graph, this is more work than is necessary. To initialize path annealing, we obtain the initial solution by conducting an A* search of the edge dual-graph. The cost of each arc in the edge dual-graph equals to the weighted Euclidean distance between its two end-points. The cost of a path is the sum of the costs of the arcs forming the path. The estimated cost to the goal is straight-line distance to the goal weighted by the map's lowest cost coefficient, μ^*. Since this is an underestimate to the goal, the search is admissible. It returns the shortest weighted midpoint path as well as the corresponding WS in which it resides. We refer to this path as the *optimal midpoint path*. Note that we should *not* expect this midpoint path to be globally optimal; nor should we expect that its corresponding WS necessarily contains a globally optimal path. However, the optimal midpoint path does represent a reasonably good initial solution.

Simulated annealing does not have to start with the optimal midpoint path. We could begin with any feasible solution regardless of its cost. So, we could overestimate distance to the goal, thereby inducing a greater depth-first component to the A* search [21]. Such an evaluation function usually finds a solution faster. The resulting midpoint path might be of higher cost, but nevertheless, will be feasible since the edge dual-graph does not pass through infinite cost regions. However, there are two advantages to initializing with admissible A* search. First, the resulting midpoint path is a reasonably good starting solution. It is an indication of a high-speed avenue with a higher potential for containing (though not necessarily) the optimal or a near-optimal path. Much empirical research indicates that simulated annealing is faster when provided with reasonably good initial solutions [4, 10, 20]. Second,

the total cost of this path provides an upper bound on the cost of the globally optimal path. Since we want the least upper bound obtainable, then it is desirable to find the best midpoint path.

6. Evaluating the Cost of a Window Sequence

In the main phase of annealing, each newly generated WS must be evaluated to find the cost of its locally optimal path. In many annealing applications the cost of a sample solution is easily evaluated. But in path annealing, cost function evaluation is difficult and may require a relatively large amount of computing time. Thus, streamlining this procedure is a high priority.

6.1 Single-Path Relaxation (SPR)

Given an arbitrary WS from start to goal, one can locate the optimal path constrained to the defining edges using the *coordinate descent* method proposed by Smith [32] (also suggested by [18]). Beginning with a path through all midpoints of the edges, we fix points on every other edge from start to goal and adjust the crossing points on those in between to their optimal locations. The next pass reverses direction (from goal to start) and works on the alternate edges. This process continues until a complete pass is made in one direction during which no point moves from its previous position more than some precision ϵ, the *relaxation tolerance*. The procedure converges on the locally optimal path because total weighted path length is a monotonically decreasing function of the sequence of adjustments [32]. It is also possible to work through the WS continuously in the same direction (such as from start to goal). However, we have found that moving in alternating directions tends to cause faster convergence.

Simulated annealing cannot afford expensive primitive operations. The coordinate descent method proposed by Smith [32] uses the *Golden Ratio* search [23] to compute the optimal crossing points, thus avoiding the direct use of Snell's Law and expensive trigonometric functions. However, simple testing of coordinate descent using Golden Ratio search suggests that Smith's method [32], though accurate, is too slow to be of practical value in path annealing, since long WS's can require a very large number of iterations. Furthermore, wide window sequences with long edges can converge very

slowly to the locally optimal path. Hence, we have designed a table-lookup technique for computing a Snell's Law crossing point in constant time. We refer to our method of coordinate descent using table-lookup as *single-path relaxation* (SPR).

Given fixed approach points P_1 and P_2, we normalize the geometry of a typical crossing episode on a standard rectangle as shown in Figure 3, where the rectangle is oriented such that the crossing edge is always parallel to the x-axis in the figure. Four parameters characterize every crossing episode geometrically similar to this rectangle:

$$\rho = \mu_2 / \mu_1 \quad \text{(ratio of cost coefficients)} \tag{1}$$

$$r = x_t / y_t \quad \text{(inverse of the slope of line } P_1\text{–}P_2\text{)} \tag{2}$$

$$y' = y_0 / y_t \quad \text{(ratio of } y \text{ coordinates of } P_0 \text{ and } P_2\text{)} \tag{3}$$

$$x' = x_0 / y_t \quad \text{(ratio of } x \text{ coordinate of } P_0 \text{ to } y \text{ coordinate of } P_2\text{)} \tag{4}$$

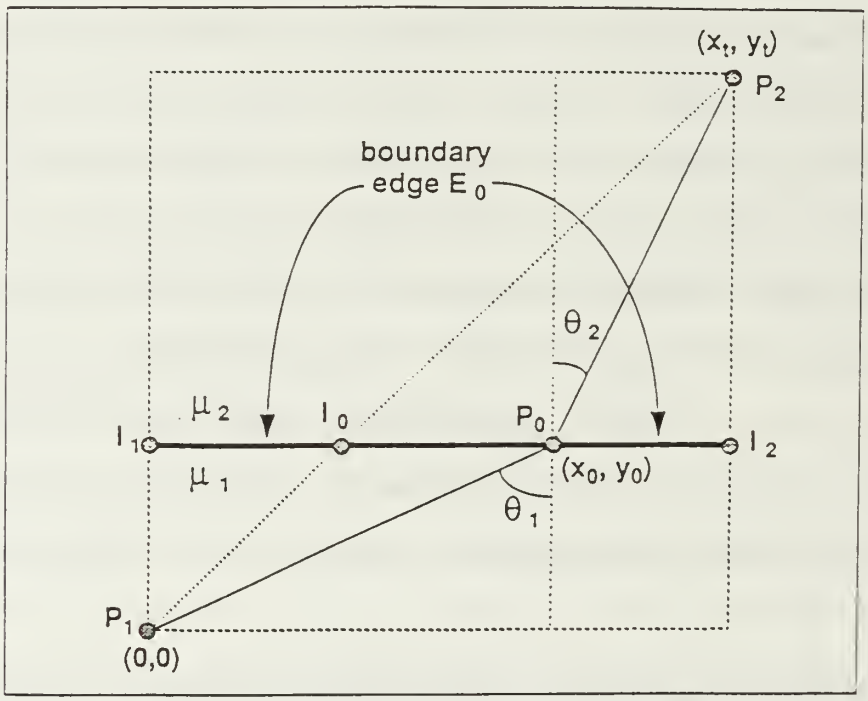

Figure 3 Crossing Episode Geometry

Given an ordered triple of ρ, r and y', the table-lookup procedure returns x' $(0 < x' < 1)$ from which we can efficiently compute the location of the desired crossing site x_0. Golden Ratio search is used to precompute the table, which is loaded at runtime during path annealing. To construct an appropriate

Snell's-Law lookup table, it is necessary to determine the expected working range of cost coefficient ratios *a priori*. By *working range*, we mean those encountered most often in maps of interest. It is not necessary for the set to include every ratio to be encountered; out-of-bounds conditions can always trigger the more expensive Golden Ratio search. However, our objective is to severely reduce the number of times a lookup must resort to iterative search. Our implementation assumes that all cross-able regions have weights drawn from the integer set $\{1, 2, \cdots, 10\}$. We use this set to indirectly index the proper value of ρ in the lookup table. Note that it is always possible to orient the geometry of an episode such that $1 \leq \rho \leq 10$ is true. Therefore, we employ a two-dimensional triangular array of 45 entries, indexed by the integer pair (μ_1, μ_2). Each entry in the array contains a plane of 4221 x' values indexed by the (r, y') pair. Figure 4 tabulates the parameter ranges and key characteristics of our prototype Snell's-Law lookup table.

	Range			
	Low	High	Increment	No. of Values
ρ	1.00	10.00	*	45
r	0.00	10.00	0.05	201
y′	0.00	1.00	0.05	21

* based on cost coefficient ratios $\mu_2/\mu_1 = \rho$
where $\mu_i \in \{1,2,3,...,10\}$ and $\mu_1 < \mu_2$

Total no. of function points = 45 * 201 * 21 = 189945

Total raw space required on disk = 760180 bytes

Figure 4 Characteristics of Snell's-Law Lookup Table

Consider the generalized crossing episode in Figure 3. From fixed points P_1 and P_2 we can easily determine intersections I_0, I_1 and I_2, and the associated distances relative to the edge, E_0. With this data the lookup procedure computes the parameters r and y'. Based on the values of μ_1, μ_2, r and y', indices are created to bracket the appropriate value for x' within a rectangle defined by four neighboring entries on the two-dimensional plane indexed by (μ_1, μ_2) in the lookup table. Simple linear interpolation determines a value for x'. This value gives the location of the optimal crossing site on E_0

relative to both y_t, and its left vertex. Since $x' = x_0/y_t$, the location of the optimal point relative to x_t is $x_0 = x'y_t$.

The location of the Snell's-Law crossing point on E_0 assumes an infinitely extended boundary edge. However, all boundary edges are actually finite length segments. Therefore, the Snell's-Law crossing point may not always lie between the vertices which define the boundary edge. In such cases, the optimal crossing point will be the vertex closest to the crossing point returned by the table-lookup procedure. We refer to these occurrences as *constrained crossings*, because the optimal crossing must violate Snell's Law (as little as possible) to satisfy restrictions of the boundary edge length. A constrained crossing is illustrated in Figure 5.

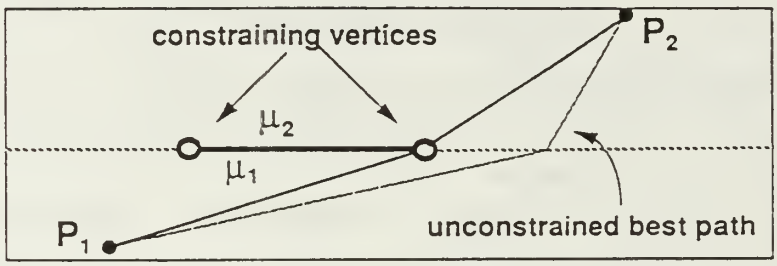

Figure 5 Constrainted Optimal Crossing

By always orienting the crossing edge E_0, fixed points P_1 and P_2, and cost coefficients μ_1 and μ_2 as shown in Figure 3, we can detect many constrained crossings before applying the complete table-lookup procedure. Note that in Figure 3 an unconstrained optimal crossing can only occur between intersection points I_0 and I_2. If both vertices of the crossing edge lie outside of the interval defined by I_0 and I_2, then a constrained crossing is certain, and table-lookup is unnecessary. The vertex closest to the I_0–I_2 interval is the optimal crossing site. Otherwise, if a table-lookup is still necessary, the location of the crossing point returned by the lookup is compared to the location of edge vertices to determine if a constrained crossing is optimal.

To compare the table-lookup procedure to pure Golden Ratio search, we generated random crossing episodes across a single boundary edge of 100 map units in length. For each episode one random point was selected from each square region on either side of the crossing edge. For the two regions, unequal pairs of weights, μ_1 and μ_2, were randomly assigned from the set $\{1, 2, \cdots, 10\}$. Timing

results of the tests indicate that the table-lookup is one order of magnitude (10 times) faster than pure Golden Ratio search. These results include time needed by the table-lookup procedure to perform occasional Golden Ratio searches when out-of-bounds conditions occur. To compute the accuracy of the lookup routine, we assume that Golden Ratio search finds the exact optimal point and, consequently, the true minimum weighted distance. Let C_{GR} be the total weighted cost of a crossing episode as computed by Golden Ratio search. Let the corresponding table-lookup value be C_{TL}. We compute relative error in weighted distance as $| C_{TL} - C_{GR} | / C_{GR}$. For 100,000 random samples the mean relative error was 3.83×10^{-7} with standard deviation 1.73×10^{-6}. The maximum relative error is on the order of 10^{-4}, and occurs in less than 0.01% of the sample. We should expect some error, since our two-dimensional linear interpolation is an approximation of a curved surface by a flat plane. Figure 6 is a plot of the discrete points in the table for $\rho = 1.1$, with the axes expanded somewhat by the integral scaling factors.

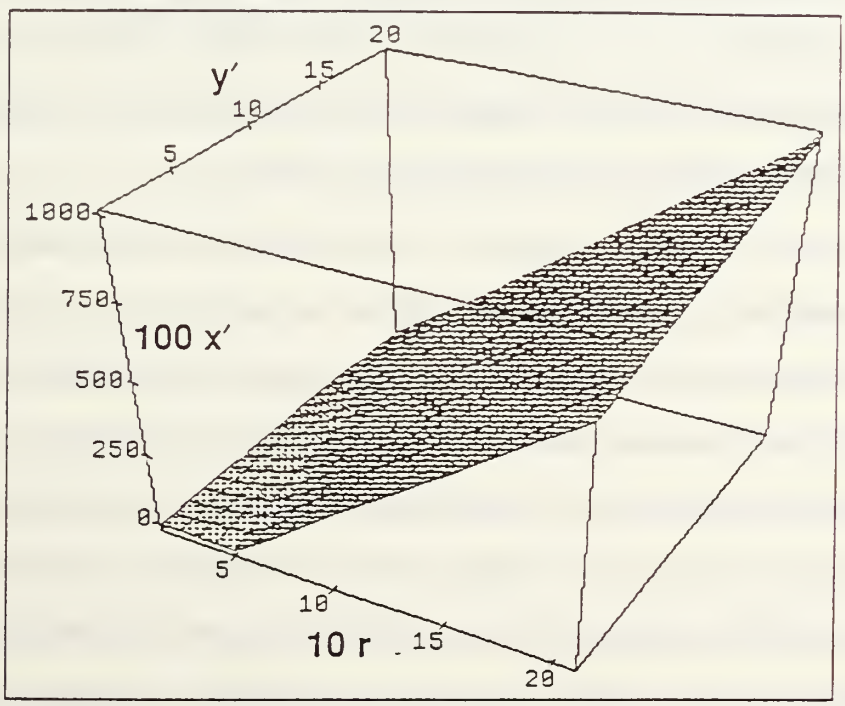

Figure 6 Surface Plot of Snell's-Law Table ($x' = f\ (r, y')$ for $\rho = 1.1$)

Although slight, the curvature of this surface is most noticeable at the lower values of r and y'. The squaring and square root operations of weighted distance soften the effect of interpolation error on x'. Nonetheless, we can rely on about 5 decimal digits of accuracy for the optimal weighted distance

of a crossing episode. Note that there are only 6 decimal digits of precision provided by our implementation language, Quintus Prolog [24]. Depending upon the length of a WS (i.e. how many crossing episodes), errors can accumulate. However, because mean relative error is essentially zero, we may assume that errors accumulated over long WS's will usually cancel one another. We conclude that the Snell's-Law table-lookup method provides single-path relaxation with a reasonably accurate and efficient means for computing optimal crossing points on boundary edges.

6.2 Uniform-Discrete-Point (UDP) Approximation

While local path optimization with SPR is fast, the procedure can sometimes run into worst-case situations. In practice we have found that even with a relatively slack relaxation tolerance, SPR may sometimes converge at an unacceptably slow rate. This happens if the midpoint path (starting condition) is far from optimal and has few sharp bends. In such situations the improvement of each crossing-point adjustment may be slight. Therefore, a large number of iterations may be necessary to optimize the path. Figure 7 illustrates a typical worst case for SPR. In this example the rate of SPR convergence decreases rapidly as the solution approaches optimum.

Usually at high annealing temperatures a large number of the WS's sampled have very large locally optimal path-costs. Therefore, accurate WS cost evaluation is only necessary when the locally optimal path might be globally optimal. Both theoretical and empirical evidence validate the use of approximations to the actual cost function [3, 33]. To capitalize on this idea, we implement a very simple approximation to pre-evaluate the cost of a WS. We refer to the method as *uniform-discrete-point* (UDP) search. It is essentially Dijkstra's algorithm restricted to a WS. UDP is used to rapidly approximate the locally optimal path through a WS, and, if necessary, to provide a starting path for SPR. Uniformly spaced points, called *edge-points*, are specified on each edge in the WS. From the start point, all paths constrained to pass through this set of points are searched. The *discrete interval*, δ, defines the upper bound for the separation distance of any pair of adjacent edge-points. Given a value for δ, all edge-points can be predetermined and stored for all boundary edges. For a given edge of length, l, the number of required edge-points is $m = \lceil (l / \delta) + 1 \rceil$. The m points (which include

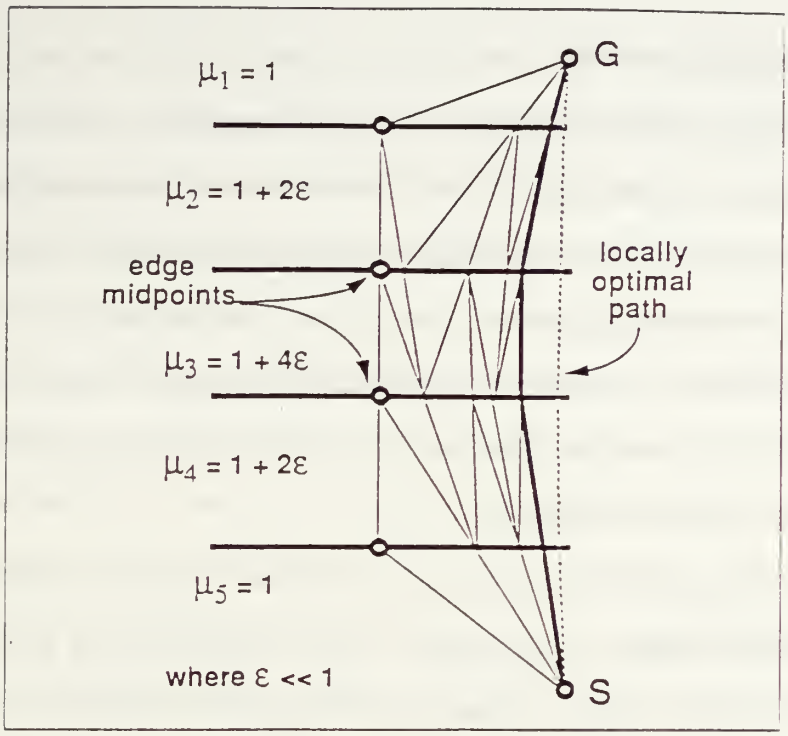

Figure 7 Worst Case Convergence for Single-Path Relaxation (SPR)
(dark path marks progress from midpoints after four passes)

terminating vertices) are then evenly spaced across the edge. For any edge with non-zero length, there must be a minimum of two edge-points - the terminating vertices. Note that the resulting structure can be viewed as a higher resolution edge dual-graph.

To approximate the optimal cost of a WS, the UDP procedure works as follows. Beginning from the start point (or inversely the goal), Dijkstra's Algorithm proceeds through the WS toward the goal. Back pointer is maintained from each edge-point to indicate the direction of the shortest path back to the start. When the goal point is reached, the shortest path is retrieved through the back pointers. This search is fast because it is restricted to a WS. Consequently, the search always proceeds in order of WS edges from the start point. The time complexity of the algorithm is $O(nm^2)$, where n is the maximum number of edges in any WS and m is the maximum number of edge-points assigned to any edge in the map. Note that m is a function of δ and the length of the longest edge in the map.

6.3 Combining UDP and SPR

In our implementation, we employ a two-stage combination of UDP approximation and SPR to find the locally optimal path in a WS. For a given WS, the UDP solution path is a more refined estimate of the local optimum than the midpoint path. Thus, UDP can hasten SPR convergence by finding a starting path which is usually closer to the true local optimum than the midpoint path. In stage one, UDP rapidly selects the locally shortest path constrained to discrete δ-spaced edge-points in the WS. In stage two, SPR begins from the path returned by UDP. Only a few passes by SPR will suffice for convergence to the true local optimum.

Both UDP search and SPR can be tuned to a desired accuracy by adjusting the discrete interval δ and the relaxation tolerance ε respectively. We have found that setting δ equal to the average length over all edges tends to balance the assignment of edge-points uniformly across the map. We assume that the length of the shortest edge is closely related to the intended map resolution and the required solution accuracy. Using mean edge-length to set δ strikes a compromise between accuracy and speed over all problem instances associated with a given map. The value of δ might also be computed dynamically based upon the lengths of boundary edges between and in the vicinity of start and goal locations. As we shall discuss in Part II of the paper, our particular choice of δ provides another opportunity to improve performance through heuristics.

6.4 Locating the Locally Optimal Paths in Reentrant Window Sequences

A *reentrant path* is one which travels from a high-cost region, to the boundary edge shared by a low-cost region, striking at the Snell's-Law critical angle, travels along the edge incurring the lower cost, and reenters, at the critical angle, the high-cost region from which it came. Such paths can be part of globally optimal solutions, particularly when start and goal lie within the same high cost region. A window sequence which contains at least one reentrant path is called a *reentrant window sequence.* Reentrant WS's present no special problem for UDP approximation, since this method is not Snell's-Law-based. However, we note that such WS's are distinctly different from those which do not contain reentrant paths. A WS which contains reentrant paths should list each reflective edge twice in

succession to indicate that there are two distinct turning points on it - the point of exit from and the point of reentry into the higher region it bounds. UDP approximation simply assumes that for each edge identifier in the WS there must be a corresponding optimal path crossing (or turning) point. Given a WS and its edge-points, the algorithm determines an optimal edge-point on each edge by running Dijkstra's algorithm. For each edge listed twice in succession, weighted distances between all pairings of its edge-points are computed using the lesser weight of the two regions separated by the edge. The pair returned as optimal on this edge is the pair along the shortest weighted path retrieved through the back pointers after the goal point is reached. If a reentrant path is, indeed, locally optimal within the WS, then the pair returned by the UDP approximation will usually be those edge-points through which a path would violate Snell's Law the least. However, in some cases a reentrant path cannot be locally optimal for the WS (i.e. when it is always cheaper for a path to travel directly through the region instead of reflecting). This happens most often when a reentrant WS is forced onto a boundary edge from its low-cost side. Figure 8 illustrates such a situation. A reentrant path against edge E_r between approach points P_1 and P_2 is not an advantage over the direct path from P_1 to P_2. In this case the pair of entry/exit points returned by the UDP approximation corresponds to a single point. The resulting path segments can always be shortcut at less cost. However, we expect this because a reentrant WS intentionally forces the cost evaluation mechanism to find the best path which hits the reentrant edge whether or not this path represents cost improvement over the original direct route. These reentrant WS's, though costing more than the original direct routes, are needed because they serve as stepping stones for annealing to reach other potentially optimal WS's. (Refer to Section 7.4 for details.)

Unlike the UDP approximation, SPR must recognize each reentrant path location within the WS because it must adjust entry and exit points to the Snell's-Law critical angle. This angle is measured from the approach points, i.e. the edge-crossing episodes just prior to and after the reflective edge. When a reentrant path is encountered during a SPR pass, critical-angle reflection points are determined using Snell's Law from the most current approach points. The stopping condition for SPR, i.e. one complete pass through the WS without any point adjustment greater than tolerance ε, depends only on crossing point adjustments and never on reflection point adjustments. This is because reflection points

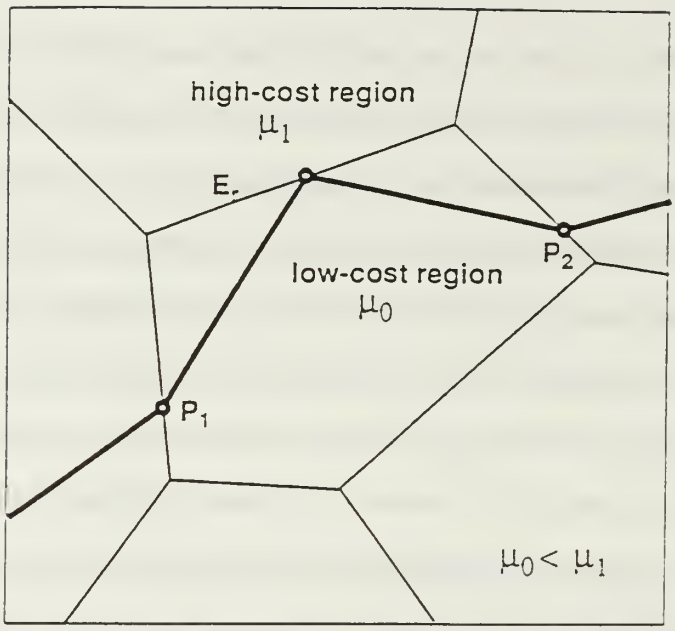

Figure 8 Uniform-Discrete-Point (UDP) Approximation is Forced to Reenter Low-Cost Region from the Low-Cost Side of a Boundary Edge

are predetermined by the adjustments of their corresponding approach points. Therefore, it is sufficient to monitor only the adjustments of crossing points (i.e. non-reflection points) in order to detect convergence to the locally optimal path.

Recall that phantom edges are edges which bound regions with equal cost coefficients. They are used to ensure that all regions are convex. When a reentrant path is forced against either a phantom edge or an edge shared by a higher-cost region, it is known *a priori* that the reentering path must violate Snell's Law and cannot be locally optimal. In such cases, SPR ignores the reentrant path route and optimizes the straight-line path between crossing episodes immediately before and after the points of reflection (i.e. approach points), effectively bypassing the reflective edge. This situation is illustrated in Figure 9. The WS still lists the reflective edge twice. The reason for this is that there may still exist an edge within the adjacent region bounded by the reflective edge on which reflection is advantageous. This will be discussed more in the Section 7.4.

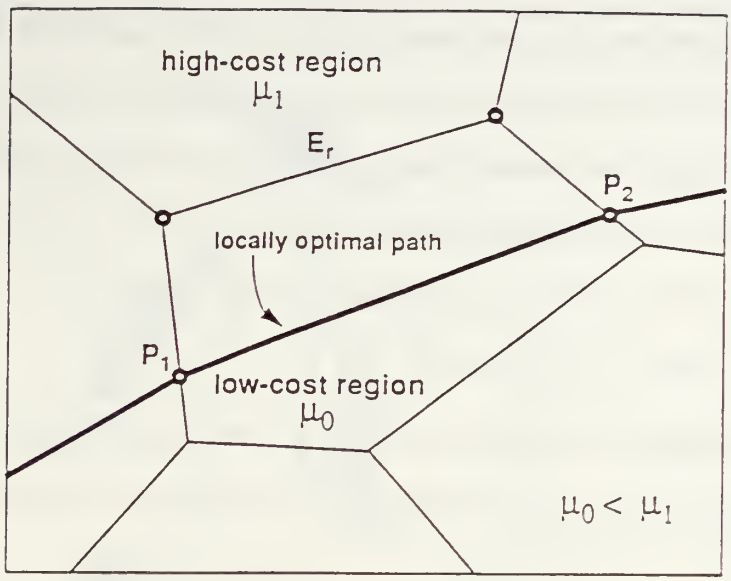

Figure 9 Single-Path Relaxation (SPR) Ignores Non-Advantageous Reentrant Path When Forced to Reenter a Low-Cost Region

7. Move Operations

Simulated annealing samples many different solutions by randomly applying move operators to explore the solution space. In path annealing, the *move generator* works in the space of WS's. Recall that we represent each WS uniquely as an ordered list of edge identifiers. A new WS can be created from another by little more than random number generation, data retrieval, and list processing. The move generator perturbs the current WS by rerouting it through neighboring edges and regions. Repetition of this process enables random search. Annealing does not normally search the entire solution space. Nevertheless, all feasible solutions should have some chance of being sampled. Our prototype employs two very simple move operators. In the following subsections, we introduce these move operators and discuss their sampling capabilities.

7.1 Vertex Rotation

Consider an arbitrary window sequence, WS_1, between designated start and goal. One of the simplest ways to obtain a new window sequence, WS_2, is to shift WS_1 across a vertex which defines one of its edges. We call this operation *vertex rotation*. The operation is relatively efficient because vertex edge-lists (see Section 4) are readily available in sorted order.

Given WS_1, path annealing generates a neighboring WS_2 as follows. Randomly select a vertex that is adjacent to one of the edges in WS_1. This vertex, called the *rotation vertex*, defines the site of intended displacement. If the rotation vertex is a border vertex then backtrack and randomly select another vertex. Let ξ be the vertex edge-list for the rotation vertex. Let ξ' be those edges in ξ which are also members of WS_1. Then, WS_2 is defined by the set operations

$$WS_2 = (WS_1 - \xi') \cup (\xi - \xi') \tag{5}$$

Eq 5 simply implies that we replace the edges in $\xi \cap WS_1$ with the edges in ξ which are not in WS_1. The result is WS_2. Of course, order in each set is significant, and list reversals must be performed as necessary.

The vertex rotation operator has an important side-effect. Given the rotation vertex V_R, the current WS is scanned for the first and last occurrences of edges in the vertex edge-list ξ of V_R. All edges in the WS between these two edges inclusive (including those which are not in ξ) will be removed. Thus, vertex rotation automatically eliminates any WS that crosses itself. This side-effect is very desirable since any WS that crosses itself cannot contain globally optimal paths.

7.2 Obstacle Jumping

The presence of obstacles necessitates special consideration. Recall that the edge dual-graph which represents the feasible solution space does not contain arcs through obstacle regions. However, we can view obstacles as enlarged vertices with non-zero area. Thus, to enable obstacle rotation or *obstacle jumping*, we employ a special data structure similar to a vertex edge-list. An obstacle edge-list (see Section 4) records the clockwise ordered list of incident edges around an obstacle. If an obstacle vertex is selected as the displacement site, the WS shifts completely across the obstacle by using the operation of Eq 5, where ξ is now the appropriate obstacle edge-list.

It is also possible to jump some obstacles by a sequence of normal interior vertex rotations. A jump of this type occurs as the result of pinching a looping path that has circled the obstacle. Figure 10 illustrates a path within a WS with the potential for jumping obstacle region A.

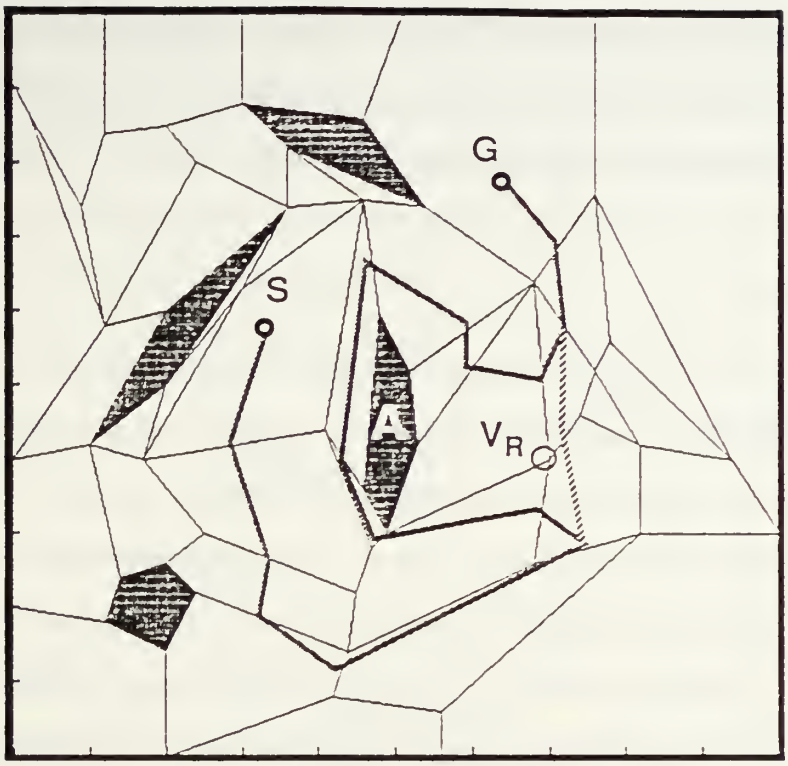

Figure 10 A Path with the Potential to Jump Obstacle A on a Single Vertex Rotation at V_R

Closing the base of the loop by rotating across vertex V_R will remove the detour around obstacle region A, replacing it with the dotted path segment and its associated WS. Not all obstacles can be jumped in this manner. For such a jump to occur enough interior vertices and edges must surround the obstacle so that a WS can be perturbed completely around it in either direction.

While obstacles and interior vertices are logically equivalent with respect to the rotation operator, we ensure that they are distinguishable. In particular problem instances it may be more efficient to reduce or temporarily prevent the capability to jump obstacles. This control is necessary because the move generator can have a natural bias toward selection and jumping of large obstacles with many vertices. This is especially true when the current WS includes several edges incident to the same obstacle, as is often the case immediately after jumping it. In such situations, given an equal probability of selecting any edge in the WS from which the rotation vertex is obtained, the vertex (or obstacle) with the most incident edges in the WS will have the greatest likelihood of being selected. This bias can waste valuable sampling time with useless repetition as the move generator attempts to jump back and

forth across a single large obstacle. As well, there are good reasons for ignoring this bias and allowing the annealing process to make its own decisions. We defer a more complete discussion of these reasons and methods of obstacle control to Part II of the paper.

7.3 Reachability

Recall that the move operators should guarantee *reachability*; that is, that all feasible solutions which could be optimal are accessible to the annealing process with a non-zero probability. This is to ensure that optimal solution cannot be overlooked intentionally. A move generator based only upon the rotation operators (vertex and obstacle) can only guarantee reachability for a subset of those WS's with the potential to contain a globally optimal path. We refer to this subset as the class of *proper window sequences*. A window sequence is *proper* if it does not enter the same cost region more than once. This definition implies that no proper WS can cross any boundary edge more than once, otherwise it necessarily reenters the same region. In contrast, an improper window sequence may enter the same cost region multiple times. It can be shown, by induction on the number of regions, that all proper WS's are reachable from any initial proper WS in a convex weighted-region map [12]. Thus, if only proper WS's could contain globally optimal paths then our move generator would guarantee reachability. Unfortunately, in some problem instances an improper WS may contain the optimal path. The next section discusses these improper WS's and the additional move operator required to reach them.

7.4 Reentrant Installation

Although an optimal path cannot cross itself, it can cross the same edge more than once, and therefore, can also reenter a region. Recall that an optimal reentrant path will obey Snell's Law by striking a boundary edge at its critical angle and traveling along the edge just inside the lower cost region. Since we disallow linear regions (e.g. roads), such a path may exit the edge in only one of two ways. It may travel to a terminating vertex (Figure 11), where Snell's Law does not apply. Or, it may reenter the region at the Snell's-Law critical angle (Figure 12). In either case, the path can still be

globally optimal. The former case involving the terminating vertex is contained within a proper WS. Since this path does not reenter the region, it must cross a different edge when it reaches the vertex. Such paths are contained within proper WS's which can be reached by the rotation operators. However, the reentrant path in the latter case must reside within an improper WS. The reason is that the path reenters a region, and thus, by definition (see Section 7.3) can only be within an improper WS.

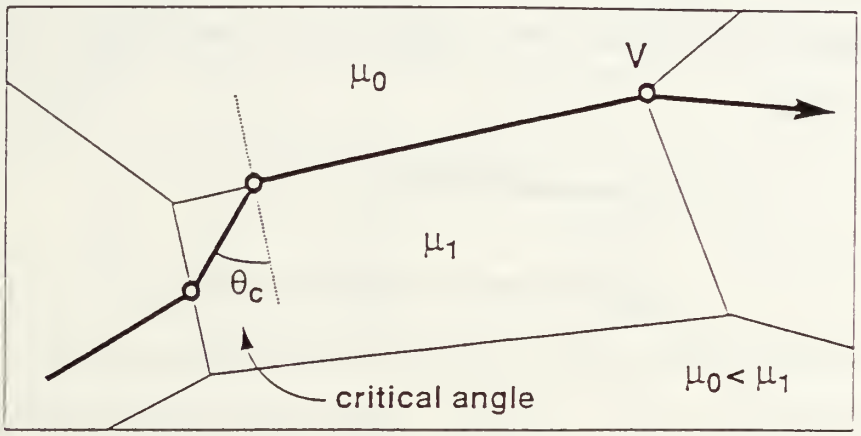

Figure 11 Critical-Angle Reflection Path (non-reentrant)

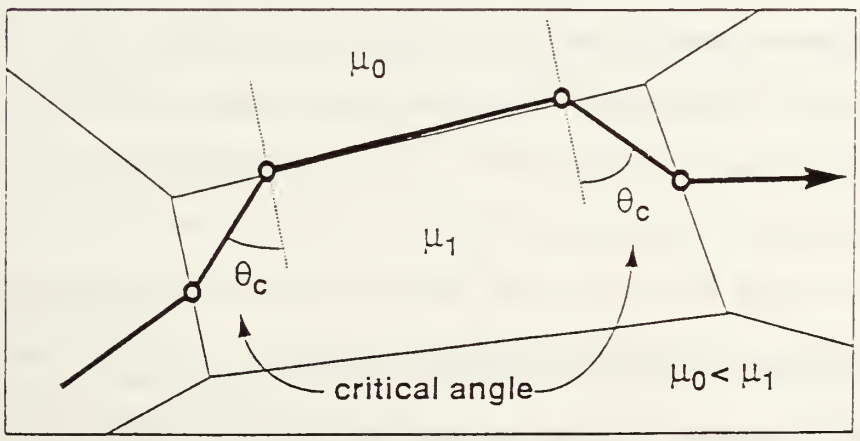

Figure 12 Critical-Angle Reentrant Path

The rotation operator alone cannot reach improper WS's since only edges that are not in the current WS can be inserted into the new WS by the rotation operator and at most one copy of each edge can be inserted into the new WS in a single rotation. We need an additional operator to reach improper reentrant WS's.

The *reentrant installation* operator perturbs a WS by forcing it to reflect on one edge. This so-called *reflective edge* is selected randomly from the set of all crossable edges that are not in the current WS but bound regions through which the current WS passes. Transformation to the new WS is a simple double insertion of the reflective edge identifier. In Figure 13, the window sequence $\{S, 1, 2, 3, G\}$ becomes $\{S, 1, 4, 4, 2, 3, G\}$ after reentrant installation on reflective edge 4. Note that the new WS is improper because by crossing edge 4 twice to reflect, the WS enters the same cost region twice.

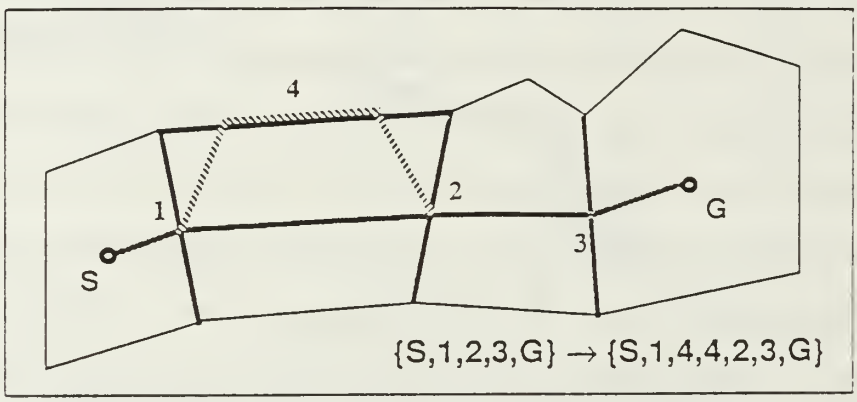

Figure 13 Installation of a Reentrant Window Sequence

It is not possible for an optimal path to reflect on two adjacent boundary edges and reenter the same region twice in a roll. Instead, the path must detour around the common vertex or cut straight across the region between the approach points. This fact is based upon a proof given in [18] which concludes that between any critical point of entry (exit) and the next critical point of exit (entry) the path must pass through at least one vertex. Because of this, our path annealing prototype will not create WS's which contain paths with consecutive reflections on two adjacent boundary edges.

It is possible for optimal paths to cross a sequence of several boundary edges, reflect and reenter a region at the last edge, and double back through the same sequence of edges in reverse order. Figure 14 illustrates the case of an optimal path which must cross several edges to reflect on the last and return.

To reach the improper WS containing this path, four reentrant installations must be *stacked*. To reach such special cases requires that the move generator install reentrant paths against an edge on which a reentrant already resides. Therefore, we allow reentrant installations to push (i.e. stack)

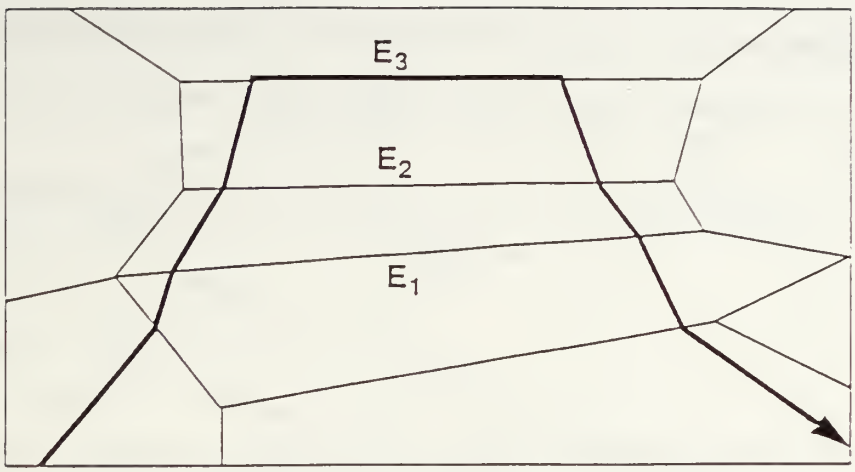

Figure 14 Reentrant Path Crosses Edges Multiple Times

through one another, ensuring that such special situations are reachable.

Recall that reentrant-path optimal solutions tend to be rare because they require particular cost-region placements and angular alignments [18, 25]. Regardless of this, we give the reentrant installation and rotation operators equal probability of selection, and let the annealing process accept or reject their resulting transitions stochastically. This means that reentrant edges will be frequently installed into the current WS. A newly installed reentrant edge which greatly improves local optimal path cost of the current WS is more likely to be kept within the current WS. This reduces the chance of eliminating good reentrant paths by the vertex rotation.

7.5 Improving Reachability

The rotation and reentrant operators described so far are capable of reaching all proper WS's and improper WS's which contain at most one reentrant path per edge. However, there still exist acyclic improper WS's which cannot be reached by these operators. Furthermore, some of these WS's can contain an optimal path. We can trace the problem to an interaction between the rotation and reentrant operators. Consider the special case of an optimal path in Figure 15. Note that the very long edge bounding the top of the narrow cost-region is crossed several times by the optimal path. The vertex rotation operator cannot create such a WS because it automatically removes edges which are already in the WS. The reentrant operator only stacks reentrants on the same edge. It cannot install them side-by-

side. No combination of these operators can generate this WS from an arbitrary proper WS. Therefore, the rotation and reentrant operators cannot ensure complete reachability for all possible globally optimal paths.

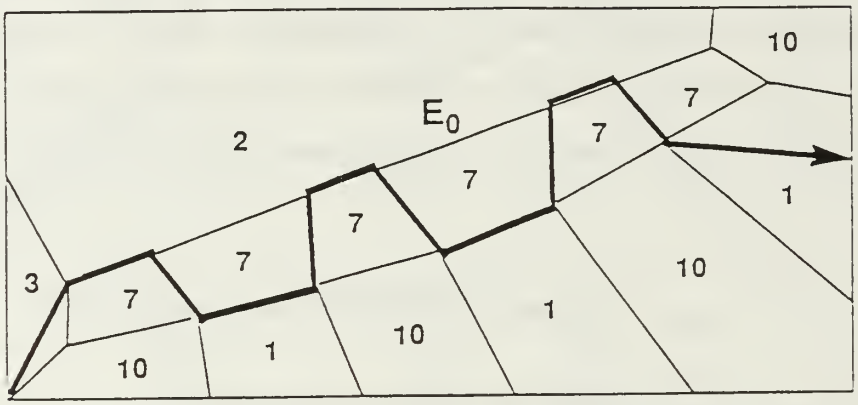

Figure 15 Optimal Path Crosses Edge E_0 Multiple Times
(integers are cost coefficients)

In the case of Figure 15, it is possible to modify the reentrant operator so that reentrants can be installed side-by-side on particularly long edges. This would require that the algorithm recognize boundary edges whose length and geometry could allow multiple reentrant optimal paths. Then, when such an edge were selected for reentrant installation, if a reentrant path already existed on this edge, then an additional decision (possibly random) would have to made to stack the reentrant or install it side-by-side.

One way to avoid introducing another more expensive move operator is to fix the source of the problem : disproportionally long edges. When a long boundary edge is situated among relatively short edges there is an increased likelihood that an optimal path may cross this edge multiple times. It is tempting to believe that complete triangulation of all regions will preclude any optimal path from crossing a single edge more than twice. Unfortunately, this is not true. Consider Figure 16 in which two near-degenerate triangular regions are adjacent with cost coefficients as shown. The path from S to G crosses the long edge (center) three times. Note that by squeezing these triangles closer to degeneracy, we can eventually reach a geometry which ensures that the path shown is optimal for S to G.

From the above simple example we see that the problem concerns resolution disparity in the map. The disproportionately long edge allows ample opportunity for short transverse crossings to greatly

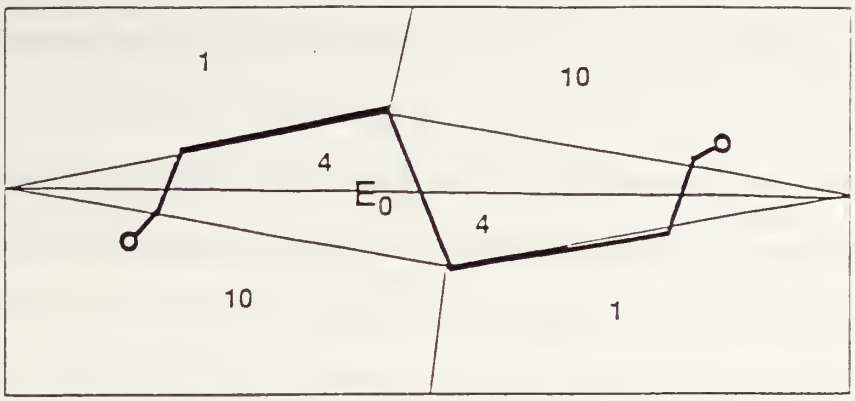

Figure 16 Optimal Path Crosses Edge E_0 Multiple Times (triangular regions)
(integers are cost coefficients)

improve the path cost. In Figure 16, two path segments that are generally parallel to the long edge become more efficient by connecting them with a very short, and therefore, relatively low-cost transverse segment. We can easily prevent this problem with the addition of one vertex at the center of the long edge and two phantom edges connecting it to existing vertices in either region. This does not change the representation of the map. It simply balances the resolution of the map in a localized area.

Some simple preliminary map analysis can reduce the risk of having optimal paths with multiple edge crossings by identifying and partitioning particularly long edges. This can be done by examining the length of each edge, E, relative to its immediate environment. Let ξ represent the set of edges bounding the two regions separated by E but not including E. Let V represent the set of vertices on the two regions. If the length of E is much greater than the average length taken over all edges in set ξ, and if the maximum distance over all perpendiculars from points in V to E is much less than the length of E, then E should be partitioned with one or more additional vertices connected to existing vertices by phantom edges. Figure 17 illustrates how the long edge in Figure 16 should be partitioned.

A comparison of cost-coefficients should also be made to ensure that the weights of the two regions separated by the edge are large relative to surrounding coefficients. If such is not the case then the possibility of a multiple crossing on the edge is minimal even if it is long.

We emphasize that the above procedure is heuristic at best. However, a completely thorough analysis is likely to be tantamount to solving many weighted-region problems. Furthermore, the rarity of reentrant-path optimal solutions and solutions which are not contained within proper WS indicates

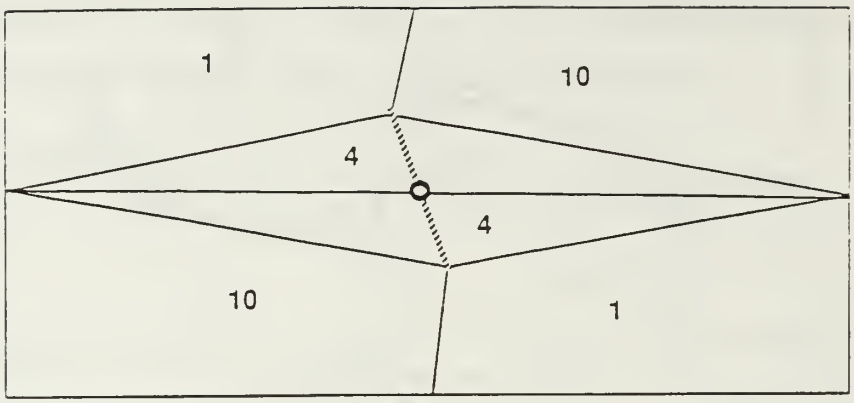

Figure 17 Introduction of Additional Edges (dotted) and Vertex (small circle) to Prevent Optimal-Path Multiple Crossings on a Long Edge

that more detailed analysis is probably not worthwhile. We desire that map resolution be reasonably balanced. In the limiting case, as more vertices and phantom edges are added, the map will approach a uniform grid representation, and the advantages of path annealing may diminish. Therefore, a long edge should be partitioned only in the case that the conditions indicated earlier are extreme.

8. Annealing Schedule and Control

Although many applications of simulated annealing require large amounts of computing time to arrive at reasonably optimal solutions, we have found that path annealing performs reasonably well without the need for such. Path annealing efficiency seems to rely more on good WRP bounds and heuristics (to be discussed in Part II of the paper) than on a well-tuned annealing schedule. Our annealing schedule is admittedly crude and uses only the simpler ideas proposed by other researchers [2, 8-11, 13, 34, 35]. The five control parameters (T_0, T_f, R, L and L_s) are established as follows.

8.1 The Starting Temperature

Recall that annealing temperature, T, controls the shape of the exponential probability distribution used to accept or reject new solution samples. While all cost-decreasing changes are accepted, changes which increase the cost are accepted with probability

$$P = e^{-\Delta C/T} \tag{6}$$

where ΔC is the difference between the costs of the new and current solutions. The acceptance ratio

over a sample of transitions is estimated by [2] as

$$\chi = (m_1 + m_2 \times e^{-\overline{\Delta C^+}/T}) / (m_1 + m_2) \qquad (7)$$

where m_1 is the number of cost-decreasing transitions, m_2 is the number of cost-increasing transitions, and $\overline{\Delta C^+}$ is the mean cost increase over the m_2 cost-increasing transitions. To ensure that annealing can sample across the solution space without trapping between high-cost peaks and freezing too early, it is desirable to set T to produce an acceptance ratio close to one. Ratios of 0.90 to 0.95 are generally considered close enough [2].

To determine a good starting temperature, T_0 (initial value of T), we have tried an empirical technique suggested by [2]. Randomly generate sample transitions in the solution space without rejections. From the sample values of ΔC compute $\overline{\Delta C^+}$. Eq 7 then determines the value of T which produces the required χ.

The conclusions of [11] suggest that simple application-specific formulas can usually be derived on the basis of desired acceptance ratio and a few problem-specific parameters. Thus, another way to determine a starting temperature value is to consider the worst-case transition that can be made by the move generator in the map. For very complex problems, such an approach is not always possible. Fortunately, our representation of the WRP by window sequences allows efficient analysis of individual transitions. We desire the value of T which makes the acceptance ratio as close to one as possible. In Eq 7, let $\chi = 0.95$, and replace $\overline{\Delta C^+}$ with an estimate of the greatest cost-increasing value, $Max\ (\Delta C^+)$.

To determine the value of T for an acceptance ratio close to one, assume that the generator applies only a single operator to the current WS, then returns a new WS for cost evaluation. Since cost-increasing reentrant installations usually do not result in large cost changes (because the way SPR computes their costs) and since they are not essential to move freely around the solution space, we will consider only rotations. We wish to determine the largest possible cost-increasing transition. This transition will likely be the site of a long edge bounding two regions with a large cost-coefficient differential.

Intuitively, $Max\ (\Delta C^+)$ should depend on an edge with a large weight differential. However, dependence upon the length of that edge may not be as clear. The reason that length is a factor is that

a rotation is followed by local optimization to find cost. If this rotation has just forced the current WS across the large weight differential, then local optimization will also force the locally optimal path to use as little of the high cost region as possible. Following this, a second transition across the other vertex of the same edge gives optimization no choice but to cross through the high-cost region. While it is possible that local optimization can still dampen the cost difference of the second transition, we assume the worst case. Ignoring smaller surrounding effects, the largest change will generally occur along the entire length of the edge. The situation is depicted in Figure 18.

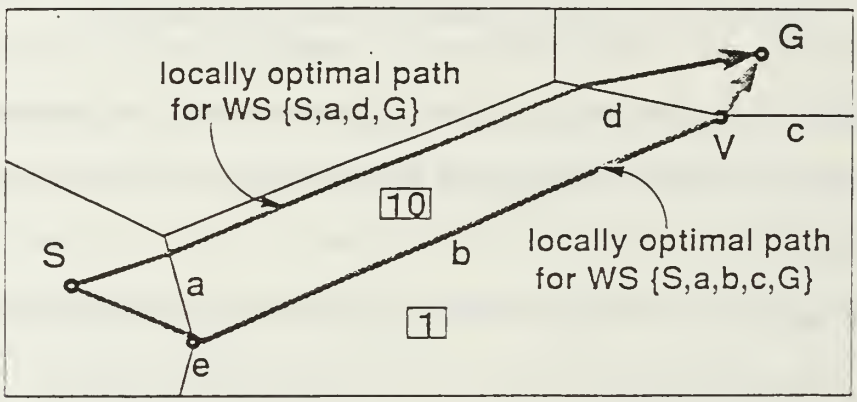

Figure 18 Maximum Cost Difference $Max\,(\Delta C^{+})$ Occurs for WS Transition $\{S,a,b,c,G\} \rightarrow \{S,a,d,G\}$ (cost coefficients are boxed integers)

Thus, to estimate $Max\,(\Delta C^{+})$, we have only to analyze the length and weighted-region cost differential for every edge in the map according to the following equation:

$$Max\,(\Delta C^{+}) = \underset{\text{over all edges } E}{Max}\ (length(E)\times(\mu_2-\mu_1)) \tag{8}$$

If we assume that in a random sample of transitions $m_1 = m_2$, then Eq 7 becomes

$$\chi = (1 + e^{-Max\,(\Delta C^{+})/T}) / 2 \tag{9}$$

Solving for T yields

$$T = -Max\,(\Delta C^{+}) / \ln(2\chi - 1) \tag{10}$$

In this approach there is no need to conduct empirical testing. One computation during map preprocessing can establish a common starting temperature for all problem instances. The major advantage here is the tendency for this method to be independent of problem instance, depending instead on the particular map. But it is conservative and can increase annealing running times unnecessarily.

However, we emphasize that this approach is quite flexible. It is possible to conduct the analysis for $Max\ (\Delta C^+)$ on a subset of map edges confined to the vicinity of start and goal. This variation is more problem-instance dependent while still very efficient.

The presence of obstacles, again, requires special consideration. Small obstacles with few boundary edges will not significantly affect transitions. However, larger obstacles can sometimes hamper lateral movement because jumping over them may require acceptance of large cost changes. If we wish to set starting temperature such that all obstacle rotations will be accepted in early annealing, it is necessary to analyze their worst-case cost differences also. Consider a single obstacle rotation. The worst transition tends to occur when the current WS is forced to detour completely around the obstacle from a single edge incident to it. To estimate the value of this worst-case cost difference, we compute cost of traveling on the boundary edges of the obstacle in a closed loop. From this we subtract the weighted length of the cheapest edge among all the boundary edges to account for that portion of the cost incurred by the original path which passed near the obstacle.

8.2 Stopping Criteria

Freezing temperature, T_f, is the final value of T at which no significant cost-increasing transition has a reasonable probability of acceptance. The reduction of unnecessary search time is as dependent upon T_f as it is on T_0. Once T is low enough, the current WS will be trapped in a local minimum from which it will likely never escape and in which it can never improve the cost beyond the lowest it has already discovered. At this point we desire that the process terminates soon. However, if it is still the case that $T \gg T_f$, then the algorithm will waste a lot of time before it halts.

How is it possible to determine the proper value of T_f when it apparently depends upon *a priori* knowledge of the optimal solution cost and its location? Fortunately, annealing runs exhibit a few similar characteristics. An annealing curve is a plot of the mean solution cost versus the logarithm of temperature. Figure 19 illustrates a curve which is typical of most annealing runs [2, 8, 13, 34, 35]. $\overline{C_T}$ is the average cost over all solutions sampled at discrete temperature T. Note that the x-axis represents $log(T)$ in the direction of time. Therefore, $log(T)$ decreases from left to right. Path-annealing curves

tend to be wavier at higher temperatures (left half of the curve) because the starting solutions reside in cost function valleys. Path annealing climbs in and out of many such valleys until the value of T is low enough to stabilize (by trapping) it within a localized area. However, at lower temperature path-annealing curves do resemble the tail in Figure 19.

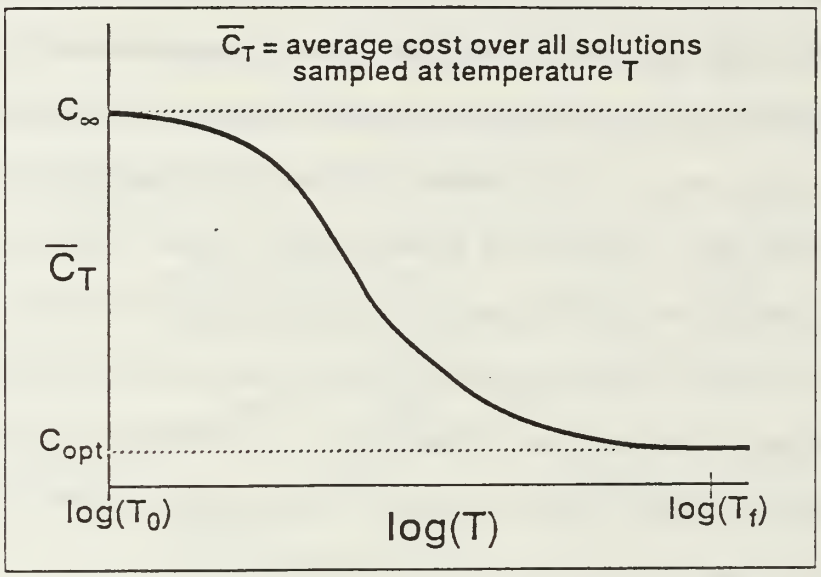

Figure 19 Annealing Curve: Average Cost at T vs. Log(T)

One way to set T_f dynamically is to extrapolate the end of the annealing curve by tracking the change in mean cost [2]. This does not work well in path annealing since, even with smoothing, the mean cost can fluctuate significantly at very low temperatures. The suggestion of [9] is simpler and a much better indicator of the freezing condition in path annealing. For a given T, if the cost difference between the largest and smallest accepted transitions is the same (or very nearly so) as the largest accepted ΔC, then apparently all solutions in the neighborhood are of very similar cost. At this point, simulated annealing should default to iterative improvement and halt. To guard against premature freezing, we suggest that the above condition exist for two or three consecutive temperatures.

The above procedure stops the annealing algorithm dynamically. Our prototype actually uses a fixed value for T_f. However, we determined its value from many observations of path annealing using the criteria of [9].

8.3 Temperature Cooling

There are several sophisticated cooling schemes which have been devised for efficient, adaptive reduction of temperature [1, 9, 14]. We prefer the more simplistic approach taken in the original work of [13], also adopted by [10]. Three fixed parameters control temperature cooling: R, L, and L_s.

The reduction or cooling factor, R (typically $0.70 \leq R \leq 0.99$), is a multiplier used to reduce T geometrically according to

$$T_{i+1} = R \times T_i \tag{11}$$

This parameter controls ΔT, the speed at which the algorithm will approach T_f. In accordance with the underlying theory of simulated annealing [13] it is best to reduce T as gradually as possible with values of R closer to one. In any case, there is a tradeoff between time spent at each temperature value and the size of ΔT. Large ΔT approaches T_f relatively fast, but requires more time at each value of T. On the other hand, smaller ΔT approaches the stopping criteria much slower, but requires less time at each T.

The amount of search time spent at each temperature is directly controlled by L. The *chain length*, L, is the maximum number of transitions attempted by the move generator at each value of T. The best value of L is generally believed to approximate the size of the local search neighborhood [2]. The local search neighborhood is the number of distinct states (i.e. window sequences) that are one transition (i.e. one application of the move generator) away from the current state. For path annealing the local neighborhood size is a function of the number of edges in a current WS, since the move generator selects transitions on this basis. However, in contrast to the cardinality of a solution set in other problems (e.g. the Traveling Salesman Problem), the number of edges in a WS changes frequently. Also note that WS length varies greatly with the number of bends. Therefore, we suggest the employment of some average or other stable value representative of WS length.

We could use the length of the starting WS found by A* search through midpoint paths. However, this WS is often biased and may not represent the size of most neighborhoods. Thus, for our testing we choose to use a fixed value for all problem instances of a map. A reasonable value for L can be derived from a map by using a sample of straight lines equal in length to the diameter of the map.

L is computed by averaging the number of boundary edges crossed by each line over all lines in the sample. This procedure is essentially an estimate of map resolution. Once again, the advantages are simplicity and problem-instance independence with an option to allow adjustments to L for particular problem instances if more efficiency is desired.

Closely associated with L is the maximum number of accepted transitions per temperature, L_s. Recall that a transition attempted by the move generator is accepted if its corresponding solution cost is less than the current solution cost, or with probability P (see Eq 6) if more. L_s is referred to as a *cutoff* for L [10]. The purpose of a cutoff is simply to compensate for overestimating the starting temperature, T_0. We desire a minimum value for T_0 at which the acceptance ratio $\chi = 0.95$ (or very close to one). It is better to begin annealing at an overestimated value of T_0, rather than risk the possibility of trapping too early in a bad local minimum. Assuming T_0 has been overestimated, until T approaches the proper value of T_0, each consecutive sequence of L_s acceptances will cause premature temperature reduction by Eq 11. L_s is essentially an annealing-specific heuristic devised by [13] to reduce search time spent at high temperatures [10]. There appears to be little published guidance as to how to establish this parameter. Some forms of the annealing algorithm do not use it. In fact, [10] claims that its effect is often insignificant, and its use is usually unnecessary if good starting temperatures can be found. We should expect some gain in efficiency by using a cutoff because our starting temperatures tend to be higher than necessary. Indeed, path annealing exhibits generally faster execution times with the same performance for $(0.67)L \leq L_s \leq (0.75)L$. Note that use of this control parameter has no effect when $L_s = L$.

9. Conclusion

In this paper, we have presented the framework for a simulated-annealing-based algorithm designed to solve large instances of the Weighted-Region Problem. We have discussed the customization of the annealing algorithm in terms of its five essential components: state space representation, initial solution, cost function evaluation, move generation and annealing schedule. Rather than just another application of simulated annealing, the proposed algorithm is an intelligent marriage of both

global and local search under probabilistic control. The major contribution of our work is not in annealing, but rather in the novelty and design of our annealing-based approach to solving the WRP. Extensive test results indicate that the new algorithm runs much faster than previous known techniques with a very minimal sacrifice in optimality. We have also developed a set of heuristics and bounding techniques to further enhance the performance and efficiency of the proposed algorithm. These techniques, together with the results of the extensive empirical study, will be presented in Part II of the paper.

References

1. E.H.L. Aarts and P.J.M. Van Laarhoven, "A New Polynomial-Time Cooling Schedule," *Proc. International Conference on Computer-Aided Design*, pp. 206-208, 1985.
2. E.H.L. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley and Sons, 1989.
3. M. Adam, O. Deutsch, and J. Harrison, "A Hierarchical Planner for Intelligent Systems," *Proc. SPIE Applications of Artificial Intelligence II*, vol. 548,, pp. 207-218, 1985.
4. E. Bonomi and J. Lutton, "The N-City Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm," *SIAM Review*, vol. 26, pp. 551-568, 1984.
5. V. Cerny, "A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *Journal of Optimization Theory and Application*, vol. 45, pp. 41-45, 1985.
6. N.E. Collins, R.W. Eglese, and B.L. Golden,, "Simulated Annealing - An Annotated Bibliography," *American Journal of Mathematical and Management Sciences*, vol. 8, pp. 209-307, 1988.
7. R.V. Denton and P.L. Froeberg, "Application of Artificial Intelligence in Automated Route Planning," *Proc. SPIE*, vol. 485, pp. 126-132, 1984.
8. B. Hajek, "A Tutorial Survey of Theory and Applications of Simulated Annealing," *Proc. 24th Conference on Design and Control*, pp. 755-760, Dec. 1985.
9. M.D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. of the International Conference on Computer-Aided Design*, pp. 381-384, IEEE, 1986.
10. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Operations Research*, vol. 37, pp. 865-892, 1989.
11. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, (Graph Coloring and Number Partitioning)," to appear in *Operations Research*, 1990.

12. M.R. Kindl, "A Stochastic Approach To Path Planning In The Weighted-Region Problem," Doctoral Dissertation, Department of Computer Science, Naval Postgraduate School, Monterey, CA, March 1991.

13. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

14. M. Lundy and A. Mees, "Convergence of an Annealing Algorithm," *Mathematical Programming*, vol. 34, pp. 111-124, 1986.

15. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, vol. 21 , pp. 1087-1092, June 1953.

16. J.S.B. Mitchell, "Planning Shortest Paths," Ph.D. Dissertation, Dept. of Operations Research, Stanford University, CA, August 1986.

17. J.S.B. Mitchell and C.H. Papadimitriou, "The Weighted Region Problem," *Proc. 3rd Annual Symposium on Computational Geometry*, pp. 30-38, 1987.

18. J.S.B. Mitchell and C.H. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision," Tech. Report No. 885, School of Operations Research and Industrial Engineering, Cornell University, January 1990. (to appear in *JACM*)

19. S. Nahar, S. Sahni, and E. Shragowitz, "Experiments with Simulated Annealing," *Proc. 22nd Design Automation Conference*, pp. 748-752, 1985.

20. S. Nahar, S. Sahni, and E. Shragowitz, "Simulated Annealing and Combinatorial Optimization," *Proc. 23rd Design Automation Conference*, pp. 293-299, 1986.

21. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.

22. F.P. Preparata and M.I. Shamose, *Computational Geometry: An Introduction*, Springer-Verlag, 1988.

23. W.H. Presse, B.P. Flannery, S.A. Teukolsky, and W.T. Wetterling, *Numerical Recipes in C*,, Press Syndicate of the University of Cambridge, 1988.

24. *Quintus Prolog Reference Manual, Release 2.5*, Quintus Computer Systems, Inc., Mountain View, CA, January 1990.

25. R.F. Richbourg, "Solving a Class of Spatial Reasoning Problems: Minimal-cost Path Planning in the Cartesian Plane," Tech. Report NPS52-87-021, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, June 1987.

26. F. Romeo and A. Sangiovanni-Vincentelli, "Probabilistic Hill-Climbing Algorithms: Properties and Applications," Memorandum No. UCB/ERL M84/34, ERL, College of Engineering, University of California, Berkeley, CA, March 1984.

27. F. Romeo, A. Sangiovanni-Vincentelli, and C. Sechen, "Research on Simulated Annealing at Berkeley," *Proc. ICCD*, pp. 652-657, 1984.

28. R.S. Ross, "Planning Minimum-Energy Paths in an Off-road Environment with Anisotropic Traversal Costs and Motion Constraints," Tech. Report NPS52-89-040, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, June 1989.

29. N.C. Rowe, "Roads, Rivers, and Obstacles: Optimal Two-Dimensional Route Planning Around Linear Features for a Mobile Agent," *International Journal of Robotics Research*, vol. 9, Dec 1990.

30. N.C. Rowe and R.F. Richbourg, "An Efficient Snell's-Law Method for Optimal-Path Planning Across Multiple Two-Dimensional Irregular Homogeneous-Cost Regions," *International Journal of Robotics Research*, vol. 9, Dec 1990.

31. N.C. Rowe, "Plan Fields and Real-World Uncertainty," *AAAI Workshop on Planning in Uncertain, Unpredictable, or Changing Environments*, Stanford, CA, March 1990.

32. T.R. Smith, G. Peng, and P. Gahinet, "A Family of Local, Asynchronous, Iterative, and Parallel Procedures for Solving the Weighted Region Least Cost Path Problem," U.C. Santa Barbara, 20 April 1988.

33. C.A. Tovey, "Simulated Simulated Annealing," *American Journal of Mathematical and Management Sciences*, vol. 8, pp. 389-407, 1988.

34. P.J.M. Van Laarhoven andE.H.L. Aarts , *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., 1987.

35. S. White, "Concepts of Scale in Simulated Annealing," *Proc. International Conference on Computer Design*, pp. 646-651, November 1984.

36. R. Wilber, "Expert Systems Aid On-Board Mission Management," *Defense Computing*, vol. 2, pp. 27-30, 1989.

Distribution List

Defense Technical Information Center,
Cameron Station,
Alexandria, VA 22304-6145 — 2 copies

Library, Code 0142
Naval Postgraduate School,
Monterey, CA 93943 — 2 copies

Center for Naval Analyses,
4401 Ford Avenue
Alexandria, VA 22302-0268 — 1 copy

Director of Research Administration,
Code 012,
Naval Postgraduate School,
Monterey, CA 93943 — 1 copy

Professor Robert B. McGhee
Code CS
Naval Postgraduate School
Monterey, California 93943-5100 — 1 copy

Dr. Man-Tak Shing
Code CS
Naval Postgraduate School
Monterey, California 93943-5100 — 10 copies

Dr. Neil C. Rowe
Code CS
Naval Postgraduate School
Monterey, California 93943-5100 — 10 copies

MAJ Mark R. Kindl
AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332 — 10 copies

Dr. Frank Pipitone
Code 5511
Naval Research Laboratory
Washington, D.C. 20375-5000 — 1 copy